Synchronous Up/Down Binary Counter for LUT FPGAs with Counting Frequency Independent of Counter Size

Alexandre F. Tenca, Miloš D. Ercegovac and Mircea R. Stan

Abstract— **The theory and practice of up-only (or down-only) prescaled counters is well understood both in industry and in the academia. Until recently it was not known if the design of a fast and large up/down binary counters would be possible. This paper presents the theory and practical application of the concept in the design of a 64-bit up/down binary counter for LUT FPGAs. The designs were implemented in Xilinx XC4000 and Atmel AT6000. The counter has a cycle time independent of the counter size. The main difference between the up-only (or down-only) counter and the up/down counter is the mechanism to obtain the next state when the counting direction changes. The counter does not have enough time to recompute the state that was valid in the previous state transition. To solve the problem, an extra register is used to store the previous state of the counter in each state transition. When the counting direction changes, the contests of the register is used as the next state. Experimental results show that the counter can scale up to hundreds of bits while keeping a short cycle time.**

Keywords— **synchronous counter, fpga**

I. Introduction

Counters are very common in many digital circuits. The basic synchronous modulo-2^n up counter structure has an incrementer and a state register. The incrementer generates the next state that is stored in the state register when a count signal (cnt) is active. The state transition implemented by an up-counter is:

$$s(t+1) = \begin{cases} s(t) & \text{if } cnt = 0 \\ (s(t)+1) \bmod 2^n & \text{if } cnt = 1 \end{cases} \quad (1)$$

where $s(t)$ is the counter state at time t.

The incrementer used to obtain $(s(t)+1) \bmod 2^n$ can be implemented in many different ways. In this paper we consider the simplest case, where the circuit is organized as a chain of Half-Adders (HA). The HA has only one gate in the path to generate the carry or sum bit. The delay of such a circuit increases linearly with the length, in bits, of the number to be incremented (in this case n). So, for large counters, the delay to generate the next state becomes unacceptable.

Ercegovac and Lang in [2] describe an implementation method that partitions a large counter into smaller ones (sub-counters). Each sub-counter has a circuit that, at the proper time, enables the sub-counter to change state. The partitioning is made such a way that the delay of the incrementer of a sub-counter is accommodated by the counting period of the sub-counter assigned to least significant bits. For example: if we have two sub-counters, one of n bits and another of m bits, such that the state is composed by the concatenation of these bits, the value represented by the most significant n bits is incremented in periods of 2^m clock cycles, when all m bits go to zero. If $n \leq 2^m$, there is sufficient time for carry propagation in the HAs' chain that generates the next state in the n-bit sub-counter, before the condition to change the state of this sub-counter is reached. We describe the partitioning algorithm later. Using this partitioning method, the cycle time of the counter can be made as low as one gate delay, independent of the length of the counter. Theoretically, the method presented in [2] has no limit (except for broadcasting of control signals).

A scheme presented by Vuillemin [11] uses the same idea of counter partitioning, but instead of using independent sub-counters, he combines the carries generated by each least significant sub-counter to obtain the count enable of the leftmost sub-counter. The length of each sub-counter is adjusted (reduced) in order to absorb the delay caused by the combination of carries. The advantage of the approach is to use fewer flip-flops – FFs – (since there is no circuit to enable the load of the next state in each sub-counter) but the cycle time is limited to at least 2 gate delays.

The previous schemes present the counter delay as a function of standard gate delays. In this work we present the delays in terms of *Function Generators* of the XC4000 LUT FPGA. We assume the reader is familiar with the organization of Xilinx FPGAs. The delays are referenced in this paper as FMAP delay for function generators in this series of devices. The characteristic values are given in the Xilinx manual [13].

The Xilinx Data Book [12] shows several counters for the XC4000 FPGAs. The fastest design uses *prescaler* technique [12], [3]. An example shows a 16-bit counter with a clock frequency of 111Mhz. The author of the design points out that the length of the counter can theoretically go up to 87 bits, with the same cycle time, but is really limited by the broadcast of control signals. The practical number of bits is 23. The counter also doesn't have a count input. The counter makes use of the fast carry logic available in the device. The area used is about 1 CLB per bit. CLB

A. F. Tenca is with the Computer Science Department, University of California, Los Angeles and University of Sao Paulo, Brazil. E-mail: tenca@cs.ucla.edu

M. D. Ercegovac is with the Computer Science Department, University of California, Los Angeles. E-mail: milos@cs.ucla.edu

Mircea R. Stan is with the with the Department of Electrical and Computer Engineering, University of Massachusetts at Amherst. E-mail: mstan@risky.ecs.umass.edu

stands for Configurable Logic Block.

All of these designs consider only the up-counting or down-counting case, and until recentely it was not known whether it is possible to design an up/down counter with the same properties of the prescaled up-counter (in [12] it was stated that "the ... prescaler technique ... cannot be used in counters that are up/down" while [11] concludes more cautiously by leaving this as an open problem).

Designs of up/down counters found in [12] show clock cycle times that increase significantly with the size of the counter. This paper presents a design of an up/down counter that has a clock cycle time independent of the counter size.

Reasons for long counters are presented in [2], [11]. This paper is organized as follows: initially, we present a short discussion of the constant time variable size up-counter proposed in [2], the next section presents an extension of the original design to allow up/down counting, and in the last section some experimental results are discussed.

II. Constant time variable size up-counter

Based on the fact that the worst case delay of a counter is caused by the incrementer circuit and that the delay is linearly related to the counter size, Ercegovac and Lang [2] proposed a design methodology for an up-counter that recursively constructs the counter by breaking it into subcounters.

The features that make this counter extremely useful are [1], [2]:

- Clock period theoretically independent of counter size. This characteristic is partially valid in practice because it depends on the synchronous paradigm for which the clock and other control signals are perfect broadcast signals.
- Can be read on the fly and the sampling rate is equal to the counting rate. These are the characteristics of a synchronous design. If these qualities are not needed (e.g. in frequency dividers) a much simpler Ripple-Counter [11], [5] can be used.
- Space complexity $O(N)$. If counters with a larger asymptotic complexity are acceptable (e.g. when N is small or when "super fast" counters are needed) ring counters or Johnson counters (Twisted Tail counters) with $O(2^N)$ space complexity can be used.
- Binary counting sequence. When this is not needed a simpler linear feedback shift register (LFSR) [6], [7] with the same $O(1)$ period and $O(N)$ space complexity but with nonbinary output sequence can be used.

Using the methodology presented in [2] a n-bit counter M is broken into M_1 (most significant) and M_2, such that M_1 is a $(n - \lceil log_2 n \rceil)$-bit counter and M_2 is a $\lceil log_2 n \rceil$-bit counter. The partitioning process repeats for M_2 and to all other modules dealing with least significant bits until a module of length one is obtained. A partitioning scheme for a 64-bit counter is shown in Figure 1.

Using this counter partitioning, M_1 has a delay that is smaller or equal to $n-1$ gate delays (chain of $n-1$ HAs), and the carry-in bit of M_1 comes in intervals greater or

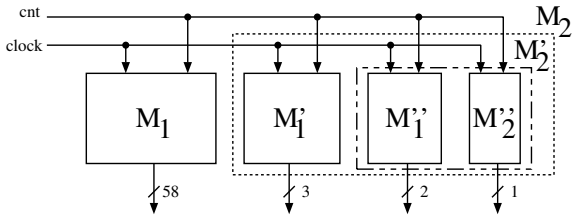


Fig. 1. Counter decomposition

equal to n clock cycles. So, if the clock cycle is made the same as a gate delay (plus some other delays: interconnection and FF delay), there is enough time for M_1 to have the incrementer stable before the carry bit arrives from M_2.

Instead of using the actual carry-out bit from M_2 (as done in [11]), M_1 has an *enable counter* that generates the enable signal to the latch that stores the counter state. The basic structure is presented in Figure 2. We use the notation $M^{k,m}$ where k is the modulo of the enable counter and m is the length of the subcounter, in bits. The enable signal to the state register of M_1 is generated in the same cycle when the delay-free carry signal from M_2 happens. As the enable counter needs to be fast, because it uses the high frequency clock of the system, a ring or a twisted-tail counter is utilized. Twisted-tail counters (Johnson) use fewer components than ring counters and are used in the implementation presented in this paper. A modulo-k Twisted-Tail Counter ($k = 2^q$) has a $k/2$-bit state vector $y(t) = (y_{k/2-1}(t), ..., y_1(t), y_0(t))$ and the following transition function: $y_i(t+1) = y_{i-1}(t)$, for $i > 0$ and $y_0(t+1) = (y_{k/2-1}(t))'$, where the prime symbol represents logic complementation. However, twisted-tail counters have the disadvantage, with respect to ring counters, of requiring extra logic to obtain the TC signal (Terminal Count).

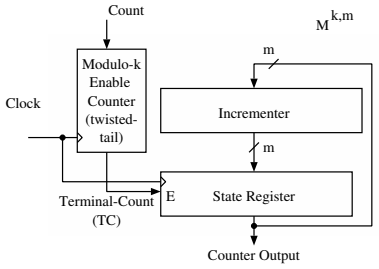


Fig. 2. Structure of a $M^{k,m}$ sub-counter

The partitioning method could be further optimized, reducing the number and size of enable counters that was obtained with the original partitioning method. This optimized partitioning method breaks a n-bit counter M into M_1, as a $(n - \lfloor log_2 n \rfloor)$-bit counter, and M_2, as a $\lfloor log_2 n \rfloor$-bit counter, whenever $(n - \lfloor log_2 n \rfloor) \leq 2^{\lfloor log_2 n \rfloor}$. When the condition does not apply, we use the original partitioning method instead. An example of this partitioning method is shown in figure 3. Notice that a smaller enable counter

is used in $M^{4,4}$ when compared to $M^{8,3}$, obtained in the the previous case.

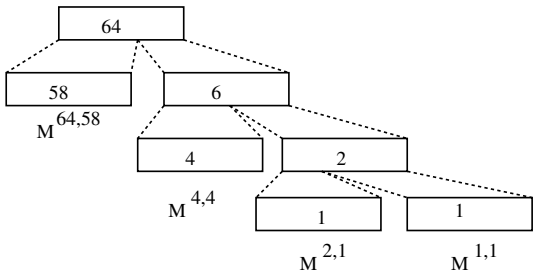


Fig. 3. optimized counter partitioning

In this counter design methodology, each sub-counter is decoupled from the others. The enable counters are synchronized and they change state at the same time. More details are presented in [2].

III. Up/Down counter design

In this section we present some modifications to allow down counting capability to the counter presented in the previous section. The design methodology proposed by Ercegovac and Lang can be modified to allow up/down counting keeping the same properties of the up counter. We cannot say he same for other up-counter designs, where the modifications would be difficult or impossible.

During normal operation, the next state is obtained incrementing or decrementing the present state of the counter. Thus, the generator of the counter next state should be capable of incrementing and decrementing the present state. The basic problem of the up/down counter is the computation of the next state when the counting direction changes. The time available to have the next counter state computed in any one of the sub-counters can be as low as one clock cycle, and it violates the assumptions in the method presented for the up-counter. The same problem happens when the counter was cleared ($s(0) = 0$), the counting direction starts as *count down* and the *cnt* input is active. To obtain the next state, it is necessary to have propagation of borrows over the length of the sub-counters, which may take more than one clock cycle.

The solution to the problem is to make the counter memorize the last state transition and recover the state when necessary, independently of the delay involved in the incrementer/decrementer circuit. Consider the counting sequence shown in figure 4, assuming that the count input is always active. We show the internal next state being computed (NS) and the present state ($s(t)$) of each subcounter. The value of NS is obtained by a combinational network and starts changing as soon as the subcounter registers a new state. The values presented in the example are arbitrary, and are used to illustrate the circuit behavior. When the direction changes, the next state computed for up-counting cannot be used. It must be obtained from the information on the previous state of the counter.

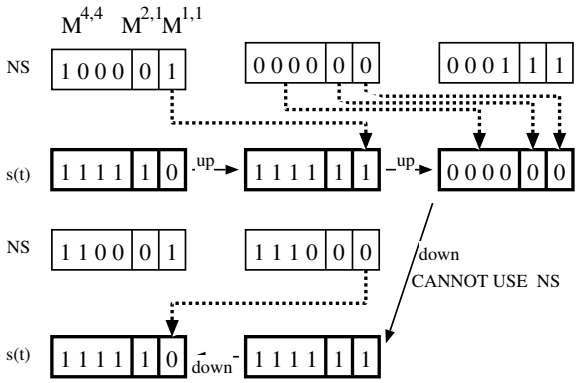


Fig. 4. Example of counting sequence with change in counting direction

Each sub-counter may need to recover the state at different instants in time, depending on the state at the time the direction changed. Considering the structure of the counter composed of sub-counters $M^{4,4}$, $M^{2,1}$ and $M^{1,1}$, and the present state is (010110), when the direction changes, each sub-counter is going to recover the previous state after 3, 1 and 1 clock cycles respectively. In order to have this feature, the twisted tail counter that generates the enable signal for the registers should be able to count up and down, what will force the inclusion of some extra circuits in the twisted tail counter to get the next state and a slightly more complex detection of the TC condition. We present in the next subsections a possible solution for these problems.

The block diagram of the proposed design is presented in figure 5. Both enable counter and next state generator modules are capable of counting up and down.

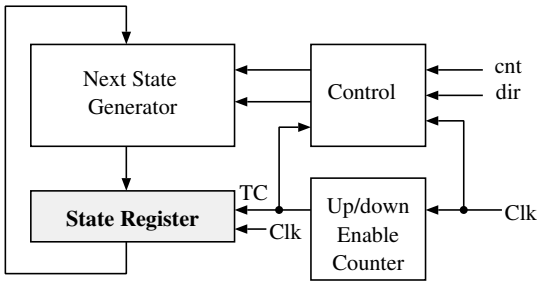


Fig. 5. Scheme of an Up/Down sub-counter

A. Up/down Enable Counter

The same way as presented for the up-counter, the up/down counter Enable Counter is implemented with a modified Twisted Tail Counter. A modulo-k twisted-tail up/down counter with $k = 6$ is shown in figure 6. Using multiplexers, the connections between the flip-flips (FF) are modified depending on the direction of counting.

The TC signal, though, must be taken from different conditions depending if the counter is counting up or down. When counting up, $TC = 1$ when the state of the enable counter is $s(t) = (100...00)$ (one state before the counter

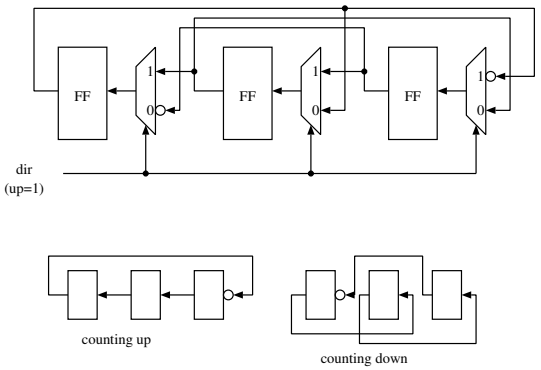


Fig. 6. Up/down twisted-tail counter

goes back to state 0) and $cnt = 1$. When counting down, $TC = 1$ when $s(t) = (000...00)$ and $cnt = 1$. The detection of zero state is done testing the extremes of the state vector. This circuit takes 1 CLB in the XC4000 FPGA, and increases the delay to generate TC, when compared to the up-counter (FHMAP delay against a FMAP delay in the original counter). The scheme is presented in figure 7.

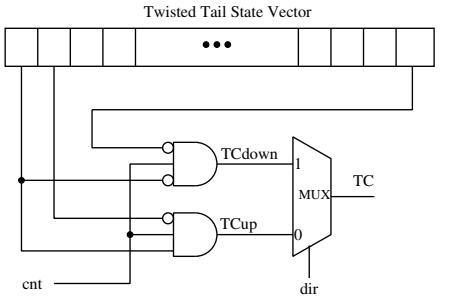


Fig. 7. Up/Down Twisted-Tail TC circuit

B. Next State Generator

The generation of the next state in normal operation is accomplished by an incrementer/decrementer circuit that can be as simple as a chain of Half Adder/Subtractors (HAS). An HAS has a control signal ($OPER$) that commands the circuit to perform addition ($a + c_{in} = 2c_{out} + s$) or subtraction ($a - c_{in} = -2c_{out} + s$). The truth table of these operations is shown below. The carry-out of one module is connected to the carry-in of the next module in the chain.

a	c_{in}	$a + c_{in}$		$a - c_{in}$	
		c_{out}	s	c_{out}	s
0	0	0	0	0	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	0	0	0

As the s output is the same for addition and subtraction, the function that generates s depends on variables a and c_{in}, but not the operation to be performed. The c_{out} output depends on 3 variables (a, c_{in} and $OPER$).

As explained earlier, there's no time to wait for the carries or borrows to propagate when the direction of counting changes. To solve the problem we we can use two different solutions that depend on the type of information that is stored. These solutions are presented in figure 8. In the first design the carry bit that was used in the last transition is used in the place of the carry computed by the HAS chain. The register was named *Carry/Borrow Register* – CBReg [10]. In the second solution the value obtained from a register with the previous state can replace the next state that is begin computed by the HASs. The register was called "shadow register" in [8]. Both designs are equivalent and can be mapped to the same resources in the FPGA chip. Lets call these registers as Previous State Registers — PSREG — from now on.

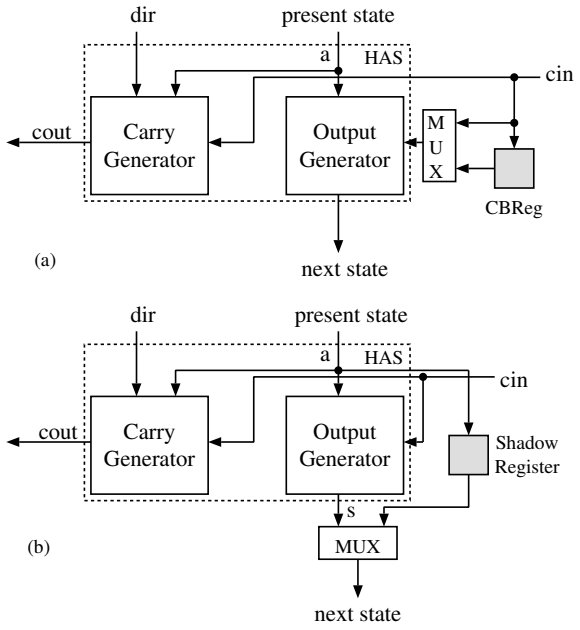


Fig. 8. Designs of the Next state generator (a) using carry/borrow from previous state transition (b) using the previous state bit

The storage of the information needed to restore the previous state of the counter will imply an increase in the number of FFs of about the length of the counter. The registers must be initialized with 1 values, to allow the condition when the counter starts with the down count direction, and the initial state is zero.

The restoration of the previous state must be controlled by each sub-counter independently. When the counter is initialized and the direction is *down* or when the direction changes during regular operation, the control circuit transmits the information of direction change and forces the MUX circuit to use the information from the PSREG to obtain the next state. The control signal that has the information on the change of direction is named in this

work as $DIRCH$. The condition is kept until a TC signal is generated by the enable counter, or the direction is restored to the previous situation. It is important to note that only the generation of the next state works based on the new direction, the carry generation continues to work in the previous counting direction ($PDIR$). That is important because the direction can change more than once during the counting period of a sub-counter (that can take many clock cycles) and the sub-counter should be able to resume the computation of the new next state that cannot be restored from the register.

A possible mapping of the HAS module, combined with the MUX module, for the case of the first design, to the XC4000 function generators is presented in figure 9, with the logical equations for each output.

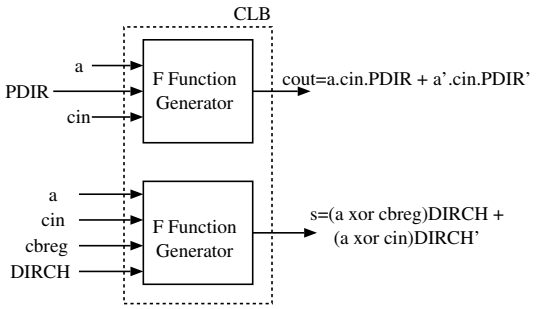


Fig. 9. Mapping of the HAS module with state recovering to FPGA Function Generators

The state register is clocked when TC is generated by the enable counter. For the first design, PSREG stores a new information only when $TC = 1$ and $DIRCH = 0$. When $DIRCH = 1$, the counting direction changed, the value in the PSREG must be kept until the next TC. This feature is necessary because the HAS chain does not have valid information on the carry/borrow lines when the state is being restored from PSREG. The counter can go back and forth between two adjacent states in the state chain of the counter with the same vector of carry/borrow bits. In the second design the registers are clocked when $TC = 1$. In both cases, if the direction signal is stable for more than one TC pulse, the sub-counter resumes regular operation.

C. Control Circuit

The control circuit is a sequential system that has the behavior represented by the state diagram in figure 10. The state changes every time TC is activated by the enable counter. The initial state S_0 is necessary for the case of counting down after clearing the counter. The output signals of the controller are: $DIRCH$ and $PDIR$. The $DIRCH$ indicates that the counting direction changed. The $PDIR$ signal indicates the previous counting direction used. The state of the circuit changes when the Enable Counter generates the TC signal.

Extra delays caused by the control logic will increase the clock cycle time of this design, when compared to the up-counter. This is discussed in the next section.

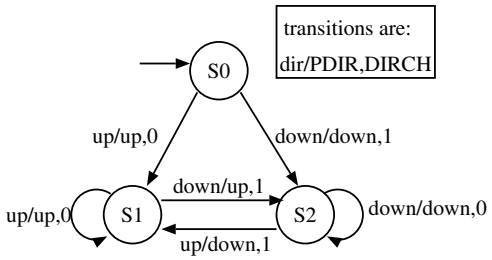


Fig. 10. Control Circuit State Diagram

IV. Experimental Results

The design using CBReg was specified in Viewlogic VHDL. The synthesis results were obtained without imposition of constraints to the synthesis tools. No manual placement or routing were performed, what leaves some space for optimizations and better performance. The design was also tested in the EVC board [4] to verify the counter operation.

We split the results in two subsections. The first one shows the implementation of the up-counter using the mentioned methodology for the XC4000 series, and the second subsection shows the implementation of the up/down counter for both XC4000 and Atmel AT6000.

A. Up-counter Implementation

The minimum clock period for the up-counter should be 1 FMAP delay plus interconnection delay and propagation time of a FF (set up time is included in the FMAP delay). This value can be made as short as 10ns. However, the implementation results shows that the broadcast of the enable signal, from the Enable Counter to the State Register, makes the propagation delay of this signal greater than 10ns. It is caused by the large fan-out of the signal and it becomes worse as we enlarge the counter size. We discuss this problem by the end of this section.

Table I shows the partitioning results obtained from the equations presented in section 2 and area estimates in terms of number of CLBs for some counter sizes. We are assuming the optimized partitioning method. The number of CLBs used is presented for two different implementations, one considering Fast Carry Logic (FCL) and the other disregarding Fast Carry Logic (w/o FCL). To implement the design we used two versions of a 64-bit counter, both described in Viewlogic VHDL. The first one uses standard structures and describes the incrementer as a chain of HAs. The second uses XBLOX add/sub module as the incrementer. The first implementation does not take advantage of the Fast Carry Logic and the second does. Because of that, the first design uses a slower incrementer. The area is almost the same for both cases.

A good implementation of the first case would give the following parameter estimates for the incrementer:

$$delay = t_p(\lceil \frac{n-4}{3} \rceil + 1) \text{ (ns)}$$

length	P1	P2	P3	P4	# FFs	#CLBs (FCL)	#CLBs (w/o FCL)
32	27	3	1	1	51	26	27
36	31	3	1	1	55	28	28
37	32	3	1	1	56	28	30
38	32	4	1	1	73	37	39
40	34	4	1	1	75	38	39
50	44	4	1	1	85	43	45
60	54	4	1	1	95	48	50
64	58	4	1	1	99	50	51
70	64	4	1	1	105	53	54
71	64	4	2	1	140	70	70
128	121	4	2	1	197	99	99

TABLE I
ESTIMATION OF THE UP-COUNTER DESIGN AREA (XC4000)

$$area = 2(\lceil \frac{n-4}{3} \rceil + 1) \text{ (CLBs)}$$

assuming a CLB delay of t_p ns (FMAP delay, interconnect delay and FF propagation time). The structure is shown in the figure 11, for an incrementer of 10 bits, using 4-input LUTs that are available in the XC4000 series. From the CLBs used in the incrementer, only $\lceil \frac{n-4}{3} \rceil$ FFs can be used by the twisted tail counter. Other FFs that are needed for the twisted tail counter will increase the area used by the final sub-counter (2 FFs per CLB). The synthesis tool used 62 CLBs.

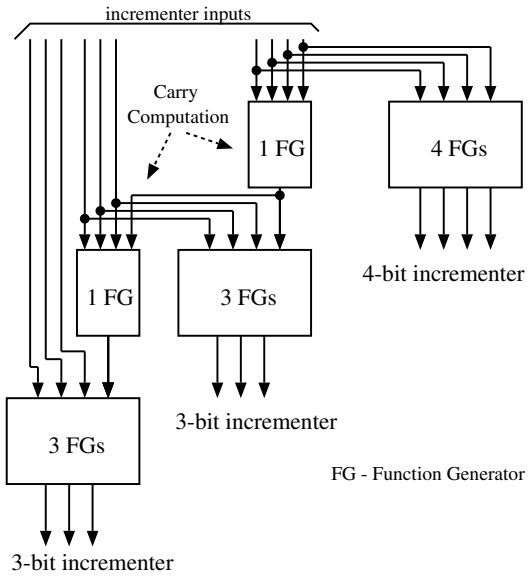


Fig. 11. Example of the incrementer circuit partitioning (10 bits), without Fast Carry Logic

In the second implementation, using Fast Carry Logic, the incrementer delay can be estimated using the equation: $8.5 + 0.75n$ (ns) (based on application notes [12]) for a XC4000-5 device. The area used by the incrementer is only $n/2$ CLBs. The total number of CLBs used in this implementation equals to half the number of FFs needed in the design (50 CLBs in a 64-bit counter).

The implementation consumes more area than the counter presented in [11]. The increase in area corresponds to the Enable Counter circuit that is used in this design.

The use of 4-input LUT FPGA technology allows the implementation of incrementers of up to 4 bits with only one CLB delay. The counter partitioning used is quite appropriate to 4-input LUTs since 3 out of 4 partitions used in the cases presented in the table have at most 4 bits (for reasonable counter size).

The most important observation is that the most significant group of bits (leftmost sub-counter) can be made larger than 2^6 bits! An incrementer of 58 bits, using fast carry logic will have a estimated delay of 52ns (without considering interconnection delays) what is far below the enable signal period of the $M^{64,58}$ sub-counter that uses it, that is $640ns$ (counter clock cycle of 10ns). Even the regular implementation using the chain of HAs would have a delay of 190ns. Based on this observation, we know that the incrementer delay is not in the critical path, and it is possible to use much larger most-significant sub-counters than was initially calculated (using the partition method). For an enable signal period of $640ns$ we could have roughly 800 bits in the leftmost sub-counter (assuming 4 sub-counters in the design). So, for all practical purposes, only 4 partitions are needed and adjustments of the counter length are done in the leftmost sub-counter, for large number of bits. These adjustments involve the width of the incrementer and state register.

As mentioned above, the propagation delay of the enable signal in the sub-counter is the critical path delay in the design. As the number of bits increase, the fan-out of the enable signal increases. The enable signal (TC of the enable counter) is generated by the combination of the enable counter state and the count signal (see Figure 7). It must be broadcast to many FFs in the leftmost sub-counter, in just one clock cycle. Our experiments show that the enable signal path delay for the 64-bit counter is 24.6ns (for the $M^{64,58}$ sub-counter).

Long lines (with smaller delay and larger fan-out capacity) can be used to minimize the delay in the path. Another approach is to split the path into a tree. The enable signal is broken into two lines, for example, using two equivalent circuits that generate TC, each circuit will feed half of the original load. When combining the idea of signal split and Long Lines to broadcast the signals, we obtained a critical path of 16.2 ns (60MHz).

Another solution is to use Global Buffers (GB). The use of GBs makes the circuit that transmits the signal less sensitive to an increase in the load. Since a small number of GBs are available, a careful placement is important to reduce interconnect delay between the signal source and the buffer.

The conclusion of this discussion is that the delay caused by the broadcast of the enable signal (TC) can be reduced independently of the sub-counter size.

need to provide more implementation data on how to solve the broadcast problem in the design!!!!!

B. Up/down Counter Implementation

For the up/down counter we implemented the designs shown in section III-B in two different technologies. The first design (using the carries) was implemented in the XC4000 and the second design was implemented in the Atmel AT6000.

B.1 using Xilinx XC4000 FPGA

In this technology, the up/down sub-counter $M^{k,m}$ consumes an area of $m + k/4 + 3$ CLBs (except for $M^{1,1}$ that uses always 0.5 CLB). The first term is the number of CLBs used by the incrementer/decrementer and registers, the second term is the area used by the twisted tail counter, and the last term is the area used by control logic.

For the 64-bit up/down counter, the area used is calculated as 91 CLBs. It represents an increase of 78% in area, when compared to the up-counter. This increase is caused by the inclusion of the extra register to restore the state.

The critical path in the up/down counter is related to the distribution of the control signals. In the path associated to the generation of DIRCH and the correct output of the next state from the incrementer/decrementer, there are 2 CLBs. The delay is 2*FMAP plus interconnect and FF delays.

If the load is excessively affecting the delay in the circuit, it's possible to reduce the load using the same ideas proposed for the up-counter design.

B.2 using the Atmel AT6000

We have implemented the design of a 64-bit up/down counter partitioned into three modules with 1, 3, 60 bits that runs at 40MHz in an Atmel FPGA (see fig. 12). The global CE (cnt) was implemented by simply gating the global clock in order to keep the overall design simple (skew problems .....??). This is the only place where the clock is gated and in case this gating is not desired it is relatively easy to implement the CE without gating the clock by slightly complicating the design. The Atmel SRAM-based AT6000 FPGAs are 2-dimensional arrays of relatively fine-grained simple logic cells which can implement a limited number of registered or combinational functions with up to 3 inputs and 2 outputs. Each cell has direct connections with its 4 neighboring cells and there are also local and express connections for routing to distant cells. The cell registers have a limited number of features and the ones that influenced the design are:

- the registers have only true (Q) outputs,
- there is no clock enable on the registers (e.g. like in the Xilinx FPGAs [12]) but registers with enable are offered as macros in the design library,
- registers only have a RESET but no SET input.

More details of this design are presented in [9].

More information on cost, comparisons?????

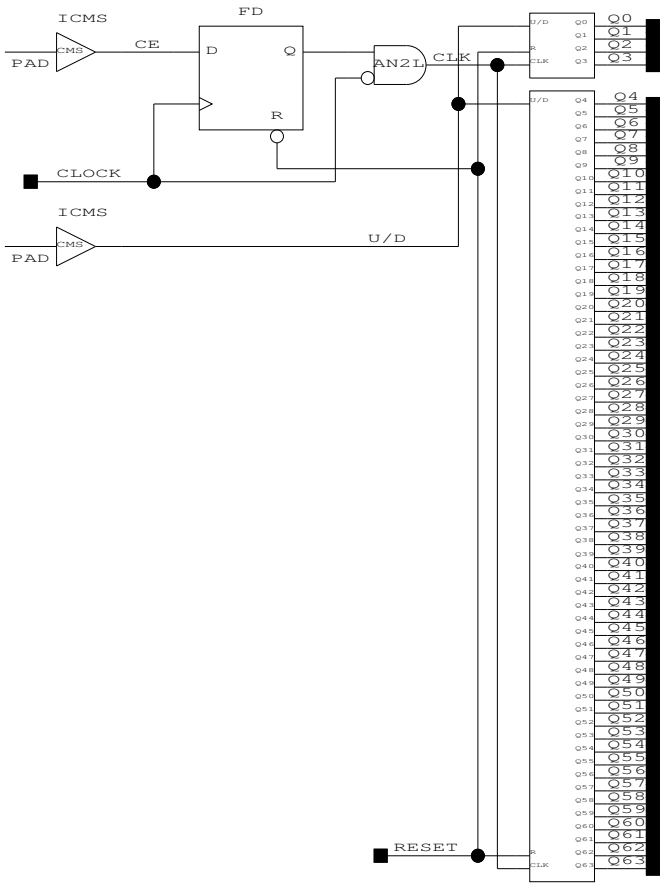


Fig. 12. Prescaled 64-bit counter. As any prescaled design this counter is partitioned into several (in this case three) submodules of exponentially increasing sizes (in this case 1, 3 and 60 bits).

V. Conclusions

We presented the methodology behind designing synchronous up/down counters of arbitrary length with period independent of counter size. We improved the functionality of a design proposed by Ercegovac and Lang for an up-counter to make it an up/down counter. The experimental results were obtained using simulation of a 64-bit counter and estimates of the area and delay for other cases.

The clock cycle time obtained for a 64-bit up-counter implemented in the XC4000 chips was 16.2ns (60MHz) but could be reduced even more, since a reasonable cycle time is around 10ns and the cycle time is independent of the counter size. The problem of broadcasting control signal was presented in the paper, and some solutions were proposed to minimize its effects.

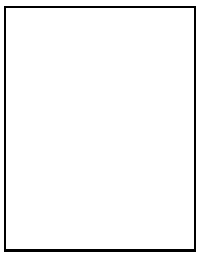
The up/down counter design was implemented in two different FPGAs for the specific case of a 64-bit up/down counter. It should be relatively easy to migrate this design to a different architecture or counter size. Since logic synthesis tools are not going to "discover" such a design, it is best to put it in a module library which can be parameterized by the counter length. Such a design only makes sense when relatively long (more than 24 bits) up/down counters are needed. For short counters better (faster or simpler) results can be probably obtained with other approaches which asymptotically are worse but are better for

small numbers.

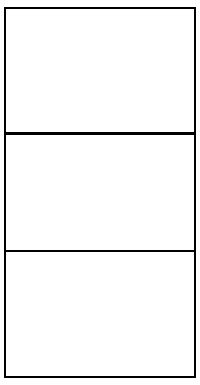
Acknowledgments. This research has been supported in part by the NSF Grant MIP-9314172 "Arithmetic Algorithms and Structures for Low-Power Systems", by CNPq and by Atmel Corp.

References

[1] Chu, D.; Phase digitizing sharpens timing measurements, *IEEE Spectrum*, July 1988, pp. 28-32.
[2] Ercegovac, M. D.; Lang, T.; Binary Counter with Counting Period of One Half Adder Independent of Counter Size;*IEEE Transactions on Circuits and Systems*, Vol. 36, No.6, 1989, pp. 924-926.
[3] Ercegovac, M. D., Lang T. and Moreno, J.; *Introduction to Digital Systems*, in preparation, John Wiley & Sons, New York, 1996.
[4] VCC– EVC1 – Engineer's Virtual Computer, User's Manual.
[5] Lutz, D. R. and Jayasimha, D. N.; Programmable Modulo-k counter, *IEEE Transactions on Circuits and Systems-Fundamental Theory and Applications*, vol.43, No.11, November 1996.
[6] Peterson, W. W.; *Error correcting codes*, MIT Press, 1961.
[7] Stan, M. R.; Shift register generators for circular FIFOs, *Electronic Engineering*, Feb. 1991, pp. 26-27.
[8] Stan, M. R. and Burleson, W. P.; Synchronous Up/Down Counter with Period Independent of Counter Size; distributed at FPGA'96.
[9] Stan, M. R. and Burleson, W. P.; Synchronous Up/Down Counter with Period Independent of Counter Size; *IEEE 13^{th} Symposium on Computer Arithmetic*;
[10] Tenca, Alexandre F., Ercegovac, M. D.;Synchronous Up/Down Binary Counter for LUT FPGAs with Counting Frequency Independent of Counter Size, *FPGA97 - ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pp. 159-165, Feb. 1997;
[11] Vuillemin, J. E.; Constant Time Arbitrary Length Synchronous Binary Counters; *IEEE 10^{th} Symposium on Computer Arithmetic*, 1991, pp. 180-183.
[12] Xilinx; *The Programmable Logic Data Book*, August 1993.
[13] Xilinx; *The XC4000 Data Book*; August 1992.



Alexandre F. Tenca is a Ph.D. student at the Computer Science Department, University of California, Los Angeles and is also affiliated to the Department of Computer Engineering and Digital Systems, University of Sao Paulo, Brazil. He obtained a M.S. degree in Computer Science in 1994 and a M.S. degree in Electrical Engineering in 1991. His research interests include computer architecture, digital design, reconfigurable architectures and computer arithmetic.



Miloš D. Ercegovac ..... your data .....

Mircea R. Stan ..... your data .....